Gröbner basis computation in Risa/Asir (Computer Algebra -- Theory and its Applications)

AUTHOR(S):

野呂, 正行

Gröbner basis computation in Risa/Asir

野呂 正行 *

Masayuki Noro

立教大学理学部

Department of Mathematics, Rikkyo University

Abstract

The first implementation of Gröbner basis computation in Risa/Asir started in 1993. The Shimoyama-Yokoyama algorithm for primary decomposition of ideals was implemented by using it in 1996. In 2003 a completely new package for Gröbner basis computation was developed. By using this new package we implemented a new primary decomposition algorithm that does not produce any redundant primary component in 2011. We want many people to use new functions, but unfortunately not a few people still use old ones. Recently we replaced the functions for bignum computation with GMP [9]. As a result we could greatly improve the efficiency of Groebner basis computation over the rationals. In this article we show the current status of Risa/Asir to make the new features known by as many people as possible.

1 はじめに

Risa/Asir におけるグレブナー基底の実装は 1992 年に始まった. その後, dp 系関数として組み込み化, F_4 の実装, nd 系関数の実装, gmp を用いた書き換えにいたるまで, さまざまな効率向上のための改良が行われてきた. 結果として, 最新版は初期のものに比べて大幅な高速化を達成したが, 残念ながら, 初期に実装された dp 系関数, およびそれを用いて実装された機能をいまだに使っているユーザが少なからず存在する. 本稿では, 開発の歴史とともに, 最新版の機能, パフォーマンスも紹介し, 新実装を使ってくれるユーザを増やす一助としたい. 特に, nd 系関数と, 新開発アルゴリズムを実装した準素イデアル分解, 根基の素イデアル分解について詳しく紹介する.

2 Risa/Asir におけるグレブナー基底計算機能

グレブナー基底計算は, Risa/Asir における主要な機能の一つである. 最初にこれまでの開発の流れの概略を述べる.

1. 最初の実装 (1992)

   Risa/Asir におけるグレブナー基底計算機能は, 下山武司による Asir ユーザ言語版が最初の実装である. 当時は多項式の内部形式としては, 再帰表現形式のみが実装されていたため, グレブナー基底計算に向く分散表現形式の単項式は, Asir 言語の配列をを用いて実装され, 多項式はそれらのリストとして表現されていた.

*noro@rikkyo.ac.jp

2. 分散表現多項式の実装 (1993)

   効率向上のため分散表現多項式を内部形式として実装した. これは DP という型であり, 現在もユーザ言語によるグレブナー基底関連計算に用いられている.

3. 部分的組み込み関数化 (1993)

   この下山版を元に, 村尾裕一により Gebauer-Moeller による useless pair 検出機能の組込み化が行われた.

4. アルゴリズム全体を組み込み化 (1995)

   これが dp_gr_main である. この時点で, 既に trace アルゴリズム [1], および斉次化との組み合わせが実装されていた. また, 有限体 F_p $(p < 2^{29})$ 上のグレブナー基底計算を行う dp_gr_mod_main も実装した.

5. F_4 アルゴリズムの実装 (1999)

   J.C. Faugère による F_4 アルゴリズム [4] の衝撃的な速度を知り, その時点での理解を元に試験的な実装を行った. これが dp_f4_main (有理数体上), dp_f4_mod_main である.

6. Weyl 代数への対応 (2000)

   Weyl 代数における Buchberger アルゴリズムを dp_weyl_gr_main として実装した.

7. 新パッケージ nd.c (2003)

   この時点までに知られていた様々な改良を取り入れた新実装を行った. これは, 分散表現多項式の新しい内部形式 ND を基礎としており, 組み込み関数名は nd で始まる. これらを nd 系関数と呼ぶ. 具体的には nd_gr nd_gr_trace, nd_f4 nd_f4_trace, nd_weyl_gr nd_weyl_gr_trace である. その後の様々な改良, 新機能追加はすべて nd 系関数に対して行われている.

8. gmp による書き換え (2018)

   これまで, Risa/Asir は独自実装の多倍長整数処理関数 (1word=32bit) を用いて諸機能を実装してきたが, 特に大きい整数の計算効率に不満があった. このため, システム全体に対し, gmp による書き換えを行った.

これらについて, 以下でより詳しく解説する.

2.1 dp 系関数について

dp 系関数は, 再帰表現と分散表現の変換, 項順序の設定, 先頭項, 基本的なものから, グレブナー基底計算を行う関数まで各種用意されている. また, これらを用いてユーザ言語で実装された機能もある. これらについて概略を述べる.

- グレブナー基底計算のための組み込み関数

  Buchberger/F_4 アルゴリズムを, 有理数体上で斉次化あり/なし, trace あり/なしで実行する dp_gr_main, dp_f4_main, 有限素体上で斉次化あり/なしで実行する dp_gr_mod_main, dp_f4_mod_main が用意されている.

- ユーザ言語で書かれたインタフェース

  gr パッケージには, グレブナー基底計算をイデアル生成系, 変数リスト, 項順序, 有限素体の標数 (有限体上の場合) のみを引数として実行する gr, hgr, gr_mod, hgr_mod が用意されている. これらは dp_gr_main, dp_gr_mod_main を適当な引数で呼び出す.

2.2 nd 系関数について

dp 系関数は, ユーザにも公開されている汎用の分散表現多項式の基本演算を用いて実装されている. 汎用性を確保するため, 単項式の指数は常に 32bit 割り当てられる. このため, たとえ 1 byte 程度で表現できる指数ばかりの場合でも, 変数が多い場合には大量のメモリを使用し. 項順序比較も多くのメモリにアクセスする必要が生ずる. また, 多項式は単項式のリストとして表現されているが, 場合によっては配列として表現する方が効率よく計算できる場合もある. このため, 汎用性を犠牲にして, グレブナー基底計算に特化した多項式の表現形式を実装し, それをもとに高速なグレブナー基底計算機能を実装したのが nd パッケージ (2003) である. そこでは, その時点までに知られていた様々な手法を実装した. 具体的には GeoBucket [3], 可変長指数ベクトル [5], 多項式表現のリスト形式と配列形式の併用, および, 次に述べるような単純化を適用した F_4 アルゴリズムの実装などである. 提供されている組み込み関数は, 有理数体, 有限体上の Buchberger/F_4 アルゴリズムを実行する nd_gr, nd_f4, nd_weyl_gr, 有理数体上で Buchberger/F_4 trace アルゴリズムを斉次化あり/なしで実行する nd_gr_trace, nd_f4_trace, nd_weyl_gr_trace (weyl は Weyl 代数での計算) である.

2.3 Risa/Asir における F_4 アルゴリズムの実装

G を中間基底とし, S を未処理の S 多項式の集合とするとき, F_4 アルゴリズム [4] の基本ステップは次のように書ける.

1 $S_{min} \leftarrow \{s \in S \mid \text{sugar(s) が最小 }\}$

2 $R \leftarrow$ SymbolicPreprocessing(S_{min},G)

3 $M \leftarrow S_{min}$ と R を行列化したもの

4 $M' \leftarrow M$ の行簡約形

5 $P \leftarrow M'$ の各列を多項式に戻したもの

6 $G \leftarrow G \cup \{p \in P \mid \mathrm{LT}(p) \notin \mathrm{LT}(R)\}$

ここで sugar(s) は全次数である. また, SymbolicPreprocessing(S_{min}, G) は, S_{min} のすべての元の G による正規形を計算できる reducer の集合 R を返す. M をガウス消去することにより, 正規形どうしの除算までまとめて遂行してしまうという点が重要である.

このアルゴリズムをそのまま実行すると, R に対応する行も簡約されてしまうことになるが, ステップ 6 により, R の要素の先頭項を持つ行は捨てられるので, この操作は無駄に見える. 実際, $|S_{min}|$ より $|R|$ が大変大きくなる場合が多く, R の行の簡約は効率に大きく影響する. これを考慮して nd パッケージでは次のような形で実装した.

1-3 前と同様

4 $M_0 \leftarrow S_{min}$ を R で簡約したものを行列化

5 $M' \leftarrow M_0$ の行簡約形

6 $G \leftarrow G \cup M'$ を多項式に戻したもの

[4] では R の簡約結果を, その後のステップの reducer として用いる (Simplify と呼ばれる) ことで, 結果として行列の非零要素が右に寄ることになり, sparse なガウス消去と組み合わせて全体として効率向上の効果が得られるとしている. これは, 有限体上の計算では妥当かもしれないが, 有理数体上では疑問に思えたので, Risa/Asir では, 別の形で実装することにした. Simplify を未実装のためこの方法の是非は不明であるが, `dp_f4_main` に比較して `nd_f4`, `nd_f4_trace` は大きく高速化した.

2.4 gmp による書き換え

Risa/Asir においては, 1990 年ごろ独自開発した多倍長整数演算プログラムを, すべての整数計算において用いていた. 当時は 32bit CPU が主流であり, そのプログラムも 1word=32bit に完全に依存した形で書かれていた. その後, 64bit CPU が主流になり, 多倍長整数演算ライブラリとして gmp を用いれば労せずして 64bit 演算による高速化が可能になる状況となったが, 独自開発プログラムで用いられているデータ構造に依存した部分がたくさんあったため, なかなか gmp を全面的に取り入れることができなかった. しかし, 2018 年についにその作業に着手し, なんとか完成させることができた. 以下この版を `asir2018` と呼ぶ. gmp により, 多倍長整数演算は高速化した. 例えば積で比較すると, 10^3 bit で 1.7 倍, 10^4 bit で 8 倍, 10^5 bit で 20 倍に及ぶ. 表は, `asir2000`, `asir2018` における, `dp` 系, `nd` 系関数の計算時間の比較である. カッコ内が `asir2000` での計算時間である. 項順序はすべて全次数逆辞書式順序, 時間は秒, CPU は Xeon E5-2650 で計測した. 有理数体上の計算は, すべて斉次化 trace アルゴリズムでの計算である.

	`dp_gr_mod_main`	`nd_gr`	`nd_f4`
C_7	5	1.6	0.3
C_8	220	38	5
$eco12$	1600	170	35

表 1: F_{32003} 上での計算

	`dp_gr_main`	`nd_gr_trace`	`nd_f4_trace`
C_7	89 (150)	9 (13)	3(10)
C_8	6700(—)	540(820)	100(520)
C_9	—	—	114h(—)
$eco12$	—	1560(4400)	400(1800)
$McKay$	8100(—)	180(1800)	39(82)

表 2: 有理数体上での計算

`asir2000` と `asir2018` の結果を比較すれば, `gmp` 導入による効果が分かる. また, `dp` 系と `nd` 系を比較すれば, 有限体上, 有理数体上いずれにおいても, `nd` 系関数が大きく高速化していることがわかる.

3 グレブナー基底計算の応用

グレブナー基底の応用としては理論的なものから方程式求解に至るまでさまざまなものがあるが, 本節ではイデアルの分解について述べる. イデアルの分解とは, イデアルを, いくつかのイデアルの共通部分の形で表すことを言う. 一般のイデアルに対しては準素イデアルによる分解が存在することが知られている. また, 根基イデアルは素イデアルにより分解できる. これらについて, Risa/Asir に実装されているアルゴリズムを紹介する.

3.1 イデアルの準素分解

dp 系関数を用いてさまざまなパッケージが開発された. ここでは, イデアルの準素分解について述べる. これについては, 現在, 次の 2 つのパッケージが利用可能である.

- `primdec` (下山-横山; 1996)

  Shimoyama-Yokoyama (SY) アルゴリズム [2] を dp 系関数により実装したものである.

- `noro_pd.rr` (野呂; 2011)

  野呂-川添による SYCI (SY with Colon ideal and Intermediate decomposition) アルゴリズム [7] を nd 系関数により実装したものである.

以下, これらのアルゴリズムの概略を述べる.

SY アルゴリズムは次のようなアルゴリズムである.

Algorithm 1 SY アルゴリズム

$\{P_1, \ldots, P_l\} \leftarrow I$ の極小付属素イデアル

$\sqrt{\tilde{Q}_i} = P_i (i = 1, \ldots, l)$ および $I = (\tilde{Q}_1 \cap \cdots \cap \tilde{Q}_l) \cap (I + \langle f_1^{s_1}, \ldots, f_l^{s_l} \rangle)$ を満たすような
イデアル $\tilde{Q}_i$, 多項式 $f_i^{s_i}$ を計算する.

$\tilde{Q}_i = Q_i \cap I_i' (i = 1, \ldots, l)$ を満たす準素イデアル Q_i, イデアル I_i' を計算する.

I_i', $I + \langle f_1^{s_1}, \ldots, f_l^{s_l} \rangle$ を準素分解する.

SY アルゴリズムの特徴は次のとおりである.

- 極小付属素イデアルから準素成分を計算する. このため, 幾何学的分解と呼ばれる.
- SY アルゴリズムは再帰的アルゴリズムである.
- $\tilde{Q}_i$ を分解する際, 無駄な成分が生ずる可能性がある. すなわち, 擬準素成分 Q_i' に対し $\tilde{Q}_i = Q_i \cap I_i'$ を満たす I_i' は一意ではないので, もし I_i' が小さければ, その分解は無駄な成分を含む可能性がある.
- $I + \langle f_1^{s_1}, \ldots, f_l^{s_l} \rangle$ を分解する際, 無駄な成分が生ずる可能性がある. すなわち, $f_1^{s_1}, \ldots, f_l^{s_l}$ は一意ではないので, $I + \langle f_1^{s_1}, \ldots, f_l^{s_l} \rangle$ が小さければ, その分解は無駄な成分を含む可能性がある.

これらのうち, 最後の 2 つは実際に問題になる場合があり, 後で実例を示す. これらの問題点を解決できるアルゴリズムとして, 次のようなアルゴリズムを考案した. k を体, $X = \{x_1, \ldots, x_n\}$, $R = k[X]$ とする.$Y \subset X$ に対し, $R_Y = k(Y)[X \setminus Y]$ である.

Algorithm 2 SYCI アルゴリズム (パート 1 : Ass(I) および中間分解)

$i \leftarrow 1$; $Q_0 \leftarrow R$
while $Q_i \neq I$ do
  $PL_i = \{P_{i1}, \ldots, P_{in_i}\} \leftarrow I : Q_{i-1}$ の極小付属素イデアル
  for $j = 1$ to n_i do
    $Y_{ij} \leftarrow P_{ij}$ に対する極大独立集合
    $f_{ij} \leftarrow (\bigcap_{k \neq j} P_{ik}) \setminus P_{ij}$ の要素
    $R_{ij} \leftarrow Q_{i-1} \cap ((I : f_{ij}^{\infty})R_{Y_{ij}} \cap R)$
  end for
  $Q_i \leftarrow Q_{i-1} \cap R_{i1} \cap \cdots \cap R_{in_i}$
  $i \leftarrow i + 1$
end while $QL_1 \leftarrow \{R_{11}, \ldots, R_{1n_1}\}$

Algorithm 3 SYCI アルゴリズム (パート 2 : 準素成分の計算)

for $i = 2$ to m do
  for $j = 1$ to n_i do
    $J_{ij} \leftarrow SaturatedSeparatingIdeal(R_{ij}, Q_{i-1}, P_{ij})$
    $Q_{ij} \leftarrow (R_{ij} + J_{ij})R_{Y_{ij}} \cap R$
  end for
  $QL_i \leftarrow \{Q_{i1}, \ldots, Q_{in_i}\}$
end for
return $(QL_1, \ldots, QL_m)$

ここで, $SaturatedSeparatingIdeal(I, Q, C)$ は $I = Q \cap (I + J)$ かつ $\sqrt{I : Q} = \sqrt{I + J}$ を満たすイデアル J を計算する手続きである. SYCI アルゴリズムは次のような特徴を持つ.

- PL_i は $\mathrm{Ass}(I) \setminus \cup_{j<i} PL_j$ において, 包含関係で極小なもの全体である. すなわち, PL_i は互いに共通部分を持たない. また, それらの和集合が $\mathrm{Ass}(I)$ とちょうど一致するので, 無駄な成分を一つも生成しない.
- ループで構成されており, 非再帰アルゴリズムである.
- PL_i は, 準素成分 Q_{ij} を知らなくても計算できる.
- Q_i, R_{ij} は I により一意的に計算できる. これは, これらがそれぞれ, $\mathrm{Ass}(I)$ の部分集合から決まるある乗法的集合 S により $IR_S \cap R$ と書けることから示せる.
- Q_{ij} は, R_{ij}, Q_{i-i} から $R_{ij} = Q_{i-1} \cap Q_{ij}$ を満たす準素イデアル Q_{ij} として計算される. この計算のもととなるイデアルを与えるのが $SaturatedSeparatingIdeal$ なる手続きである.
- アルゴリズムのいろいろな部分が並列化できる.

– $\sqrt{I:Q_{i-1}}$ の素イデアル分解は, $Q_{i-1}=\langle h_1,\ldots,h_s\rangle$ とすると $\sqrt{I:Q_{i-1}}=\bigcap_{k=1}^{s}\sqrt{I:h_k}$ なので, 各 $\sqrt{I:h_k}$ の素イデアル分解を行って重複を省けば $\sqrt{I:Q_{i-1}}$ の分解が得られる. $\sqrt{I:h_k}$ の素イデアル分解は独立に行うことができる.

– 各 i に対する R_{ij} $(j=1,\ldots,n_i)$ の計算, および, 全ての i,j に対する Q_{ij} はそれぞれ独立に行うことができる.

表 3 は, `primdec`, `noro_pd.rr` がそれぞれ提供する準素分解関数 `primadec`, `noro_pd.syci_dec` の計算時間の比較である. 例としては, SY に限らず, 再帰型の分解アルゴリズムで無駄な因子が出やすい単項式, 二項式, 三項式で生成されるイデアルを用いた.

	`primadec`	`noro_pd.syci_dec`	$\lvert PL_1\rvert,\lvert PL_2\rvert,\ldots$
$A_{2,3,5}$	>1h	1.5	10,5,3,1
$A_{2,4,4}$	—	4.8	15,12,4,1
$A_{2,5,5}$	—	10h	100,107,80,61,35,32,18,4,1
T_1	1100	69	49,36,26,23,17,12,5,1
T_2	error	32	15,22,15,7,4
T_3	error	110	46,68,64,19,3

表 3: 単項式, 二項式, 三項式で生成されるイデアルの準素分解 (計算時間:秒)

これらの例の実際の生成系は [7] を参照してほしい. [7] では, より多くの例に対し, Singular の SY 実装も含めた比較を行っている. 少なくともこれらの例に対しては, SYCI アルゴリズムの, 無駄な成分を生じないという性質が効率的計算に大きく貢献していると言える.

3.2 $\sqrt{I}$ の素イデアル分解

準素分解アルゴリズムにおいて重要なステップである I の極小付属素イデアル計算, すなわち $\sqrt{I}$ の素イデアル分解計算は, I の零点集合の既約分解を求める上でもそれ自身重要である. すなわち $\sqrt{I}=\cap_i P_i$ を, 素イデアル P_i による $\sqrt{I}$ の分解とすれば $V(I)=\cup_i V(P_i)$ で $V(P_i)$ は既約な代数的集合となる. この分解を, 極大独立集合を用いた 0 次元化を経由して行う方法として Laplagne [6] によるアルゴリズムが知られている [1].

Algorithm 4 Laplagne アルゴリズム ($\sqrt{I}$ の素イデアル分解)

$J\leftarrow\langle 1\rangle$; $PL\leftarrow\emptyset$
while $J\neq\sqrt{I}$ do
  $f\leftarrow J\setminus\sqrt{I}$ の要素; $I'\leftarrow I:f^{\infty}$
  $U\leftarrow I'$ に関する極大独立集合
  $C\leftarrow\sqrt{I'K(U)[X\setminus U]}$ の $K(U)[X\setminus U]$ における素イデアル分解
  $PL\leftarrow PL\cup\{P\cap K[X]\mid P\in C\}$; $J\leftarrow J\cap(\cap_{P\in C}(P\cap K[X]))$
end while
return PL

[1] [6] は根基を計算するアルゴリズムを提案している. このアルゴリズムにおいて, 0 次元化したものの根基を作る部分を素イデアル分解に変えたものが Algorithm 4 である.

このアルゴリズムにおいて, $\sqrt{I'K(U)[X \setminus U]}$ の素イデアル分解を計算する際, 重要な計算としてイデアルを法とした最小多項式の計算がある.

定義 3.1
I を, 体 K 上の多項式環 $K[x_1, \ldots, x_n]$ の 0 次元イデアルとするとき, 各 x_i に対し, $m(x_i) \in I$ が存在して $I \cap K[x_i] = \langle m(x_i) \rangle$ を満たす. $m(x_i)$ を I を法とする x_i の最小多項式とよぶ.

K が有理数体, 有限体の場合は線形代数により容易に $m(x_i)$ を求めることができるが, K が有理関数体の場合, その計算法は様々なものが考えられる. イデアル $I \subset \mathbb{Q}(Z)[X]$ が $\mathbb{Q}(Z)$ 上 0 次元とする. $I = \langle f_1, \ldots, f_m \rangle \subset \mathbb{Q}(Z)[X]$ ($f_i \in \mathbb{Q}[Z, X]$), $J = \langle f_1, \ldots, f_m \rangle \in \mathbb{Q}[Z, X]$ とおく. $\mathbb{Q}(Z)$ 上の最小多項式の計算法には次のような方法がある (他にもいろいろありうる).

方法 1. $\{x_1, \ldots, x_{n-1}\} >> \{x_n\}$ なる消去順序に関する I のグレブナー基底 G に対し, $G \cap \mathbb{Q}(Z)[x_n] = \{m(x_n)\}$.

方法 2. $\{x_1, \ldots, x_{n-1}\} >> \{x_n\} \cup Z$ なる消去順序に関する J のグレブナー基底 G_1 に対し, G_1 の要素のうち x_n に関する次数が最小のものが $m(x_n)$ である.

方法 3. Z のうち 1 変数を選んで消去してから方法 1. を適用する.

方法 4. 有限体 F_p 上で最小多項式を計算し, その係数を未定係数に換えて, J のグレブナー基底を用いてメンバーシップ問題として $m(x_n)$ を求める.

方法 5. 方法 2 において, 中国剰余定理を用いて G_1 を求める.

これらのうちどの方法が最適か, 入力から判断するのは難しい. そこで, 素イデアル分解関数 `noro_pd.prime_dec` には OpenXM による分散計算を用いた最小多項式の競争的計算が実装されている.

3.3 計算例

ここでは一つの例として, Oaku [8] による Landau-Nakanishi 多様体の既約分解について述べる. Laudau-Nakanishi 多様体は, Feynman 積分と呼ばれる hyperfunction の特異性を記述する. 次のイデアルは, 2 次元時空における ice-cream cone と呼ばれる Landau-Nakanishi 多様体の定義イデアルである.

$$
\begin{aligned}
I \;=\; & \langle a_2(k_{20}^2 - k_{21}^2 - m_2^2), a_4(k_{40}^2 - k_{41}^2 - m_4^2), \\
& a_1(y^2 + (-2k_{20} - 2k_{40})y - z^2 + (2k_{21} + 2k_{41})z + k_{20}^2 + 2k_{40}k_{20} + k_{40}^2 - k_{21}^2 - 2k_{41}k_{21} - k_{41}^2 - m_1^2), \\
& a_3(x^2 + (-2y + 2k_{40})x + y^2 - 2k_{40}y - z^2 + 2k_{41}z + k_{40}^2 - k_{41}^2 - m_3^2), -a_1y + (a_2 + a_1)k_{20} + a_1k_{40}, \\
& -a_1z + (a_2 + a_1)k_{21} + a_1k_{41}, a_3x + (-a_1 - a_3)y + a_1k_{20} + (a_4 + a_1 + a_3)k_{40}, \\
& (-a_1 - a_3)z + a_1k_{21} + (a_4 + a_1 + a_3)k_{41}, \\
& -a_1x + u + a_1y - a_1k_{40}, a_3x + v + (-a_1 - a_3)y + a_1k_{20} + (a_1 + a_3)k_{40}, \\
& w + (-a_1 - a_3)z + a_1k_{21} + (a_1 + a_3)k_{41} \rangle
\end{aligned}
$$

$\sqrt{I}$ の既約分解の成分のうち, $a_i \neq 0$ に対応するものが重要で, 計算の結果

$$
\begin{aligned}
J \;=\; & \langle k_{20}^2 - k_{21}^2 - m_2^2, k_{40}^2 - k_{41}^2 - m_4^2, \\
& y^2 + (-2k_{20} - 2k_{40})y - z^2 + (2k_{21} + 2k_{41})z + k_{20}^2 + 2k_{40}k_{20} + k_{40}^2 - k_{21}^2 - 2k_{41}k_{21} - k_{41}^2 - m_1^2,
\end{aligned}
$$

$$x^2+(-2y+2k_{40})x+y^2-2k_{40}y-z^2+2k_{41}z+k_{40}^2-k_{41}^2-m_3^2, -a_1y+(a_2+a_1)k_{20}+a_1k_{40},$$
$$-a_1z+(a_2+a_1)k_{21}+a_1k_{41}, a_3x+(-a_1-a_3)y+a_1k_{20}+(a_4+a_1+a_3)k_{40},$$
$$(-a_1-a_3)z+a_1k_{21}+(a_4+a_1+a_3)k_{41}\rangle$$

を得た. この計算において現れた最小多項式計算は 32 回で, 2 つの場合を除いて方法 1, 2, 3 が同程度効率よく計算できた. 残る 2 つの場合に対する各方法の計算時間が表 4 である. この表により, どれか 1 つの方法を選ぶと計算が滞ってしまうことが分かる.

方法	1	2	3	4	5
No. 15	0.2sec	$>$ 10min			
No. 32	$>$ 10min	2min	5min	2min	$>$10min

表 4: 最小多項式の計算

参 考 文 献

[1] C. Traverso, Gröbner trace algorithms. In Proc. ISSAC 1988, LNCS **358**,125-138 (1988).

[2] T. Shimoyama, K. Yokoyama, Localization and Primary Decomposition of Polynomial ideals. *J. Symb. Comp.* **22**, 247-277 (1996).

[3] T. Yan, The Geobucket Data Structure for Polynomials. *J. Symb. Comp* **25**, 285-293 (1998).

[4] J.-C. Faugère, A new efficient algorithm for computing Gröbner bases (F_4). *Journal of Pure and Applied Algebra* **139**, 61-88 (1999).

[5] H. Schönemann, SINGULAR in a Framework for Polynomial Computations. Algebra, Geometry and Software Systems, M. Joswig and N. Takayama (eds.), Springer, 163-176 (2003).

[6] S. Laplagne, An Algorithm for the Computation of the Radical of an ideal. In Proc. ISSAC 2006, ACM Press, 191-195 (2006).

[7] T. Kawazoe, M. Noro, Algorithms for computing primary ideal decomposition without producing intermediate redundant components. *J. Symb. Comp.* **46**, 1158-1172 (2011).

[8] T. Oaku, An attempt to compute holonomic systems for Feynman integrals in two-dimensional space-time. to appear in RIMS Kokyuroku.

[9] T. Granlund et al., GNU Multiple Precision Arithmetic Library 6.1.2, `http://swox.com/gmp/` (2016).